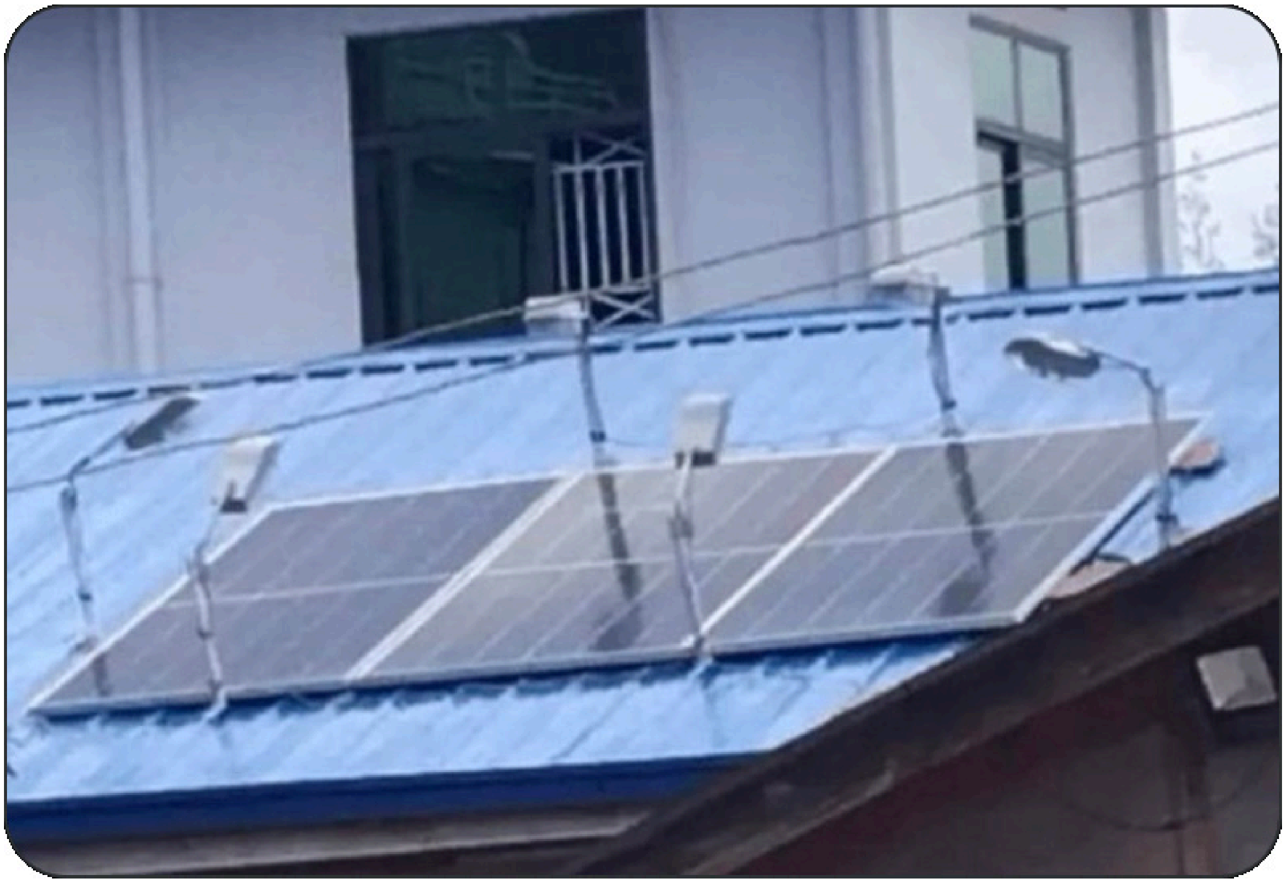
Security Now! #962 - 02-20-24
The Internet Dodged a Bullet

This week on Security Now!

What's the worst mistake that the provider of remotely accessible residential webcams could possibly make? What surprises did last week's Patch Tuesday bring? Why would any website put an upper limit on password length? And for that matter, what's up with no use of special characters? Will Canada's ban on importing the Flipper-Zero hacking gadgets reduce car theft? Exactly why **didn't** the Internet build-in security from the start? How could they miss that? Doesn't Facebook's notice of a previous password leak information? Why isn't TOTP just another password that's unknown to an attacker? Can exposing SNMP be dangerous? Why doesn't eMail's general lack of encryption and other security make eMail-only login very insecure? And, finally, what major cataclysm did the Internet just successfully dodge? And is it even possible to have a "minor cataclysm"?

Today, we'll be taking a number of deep dives after we examine a potential solution to global warming and energy production as shown in our terrific picture of the week. Some things are so obvious in retrospect.

When you're nine years old you wonder why no one ever thought of this before... Adults are so clueless!



Security News

Wyze? … not so much.

In reporting this story I'm reminded of Leo's comment about his wife Lisa "wisely" (so to speak) forbidding cameras of any kind inside the house. 9to5Mac's headline read: *"Wyze camera breach let 13,000 customers view other people's homes"*. Tom's Hardware: *"Wyze security failure let 13,000 customers see into other users' homes"*. GeekWire: *"Wyze security cam incident that exposed images to other users impacts more than 13,000 customers"*. And BleepingComputer wrote: *"Wyze camera glitch gave 13,000 users a peek into other homes"*. One of our listeners, a user of Wyze monitoring cameras, was kind enough to share the entire eMail he received from Wyze. But BleepingComputer's coverage included all of the salient details and added a bit more color, as you might expect. Here's what Bleeping just wrote yesterday:

Wyze shared more details on a security incident that impacted thousands of users on Friday and said that at least 13,000 customers could get a peek into other users' homes. The company blames a third-party caching client library recently added to its systems, which had problems dealing with a large number of cameras that came online all at once after a widespread Friday outage.

Multiple customers have been reporting seeing other users' video feeds under the Events tab in the app since Friday, with some even advising other customers to turn off the cameras until these ongoing issues are fixed.

Wyze wrote: "The outage originated from our partner AWS and took down Wyze devices for several hours early Friday morning. If you tried to view live cameras or events during that time you likely weren't able to. We're very sorry for the frustration and confusion this caused. As we worked to bring cameras back online, we experienced a security issue. Some users reported seeing the wrong thumbnails and Event Videos in their Events tab. We immediately removed access to the Events tab and started an investigation."

Wyze says this happened because of the sudden increased demand and led to the mixing of device IDs and user ID mappings, causing the erroneous connection of certain data with incorrect user accounts. As a result, customers could see other people's video feed thumbnails and, in some cases, even video footage after tapping the camera thumbnails in the Wyze app's Events tab.

In emails sent to affected users, Wyze confessed: "We can now confirm that as cameras were coming back online, about 13,000 Wyze users received thumbnails from cameras that were not their own and 1,504 users tapped on them. We've identified your Wyze account as one that was affected. This means that thumbnails from your Events were visible in another Wyze user's account and that a thumbnail was tapped. Most taps enlarged the thumbnail, but in some cases it could have caused an Event Video to be viewed."

Wyze has yet to share the exact number of users who had their video surveillance feeds exposed in the incident. The company has now added an extra layer of verification for users who want to access video content via the Events tab to ensure that this issue will not happen in the future. Additionally, it adjusted systems to avoid caching during user-device relationship checks until it can switch to a new client library capable of working correctly during "extreme events" like the Friday outage.

I like Wyze, and their cameras are adorable little beautifully designed cubes – they look like something Apple would produce. But at this stage in the evolution of our understanding of how to do security, I cannot imagine streaming to the cloud, cameras that are looking into the interior of my home. The cloud and real time images of the contents of my home do not mix. I understand that for most users the convenience of being able to login to a Wyze camera monitoring website to remotely view one's residential video cameras is difficult to pass up. And the whole thing operates with almost no setup. And Wyze's response to this incident appears to be everything one could want. The letter that our listener shared, unlike the letter that Bleeping-Computer quoted, said that his account had not been affected. So Wyze has been on the ball.

I should add, however, that I'm a bit skeptical about the idea that simply overloading a caching system could cause it to get its pointers scrambled and its wires crossed, thus causing it to get its users confused. If it really did happen that way, it was a crappy cache.

Patch Tuesday

We haven't checked in on a Microsoft Patch Tuesday for a while. Last week's update was notable for its quiet inclusion of a mitigation for what the industry's DNS server vendors have described as ***"the worst attack on DNS ever discovered."*** It seems to have slipped under much of the industry's radar, but not this podcast's. Since it could have been used to bring down the entire Internet, it's today's topic, which we will, of course, be getting back to later.

In the meantime the Internet is still here, so Microsoft doesn't get off the hook for fixing another round of problems with its products. Last week's patches included fixes for a pair of 0-days that were being actively exploited even though they were not the worst from a CVSS rating standpoint; those were a different pair of 9.8's.

Overall, Microsoft released patches to repair a total of 73 flaws across its product lineup. Of those 73, 5 are Critical, 65 are Important, and the last 3 have Moderate severity. Fixed separately, were the 24 flaws that were repaired in Edge since the release of last month's updates.

The two 0-days are a Windows SmartScreen Security Feature Bypass carrying a CVSS of 7.6 and an Internet Shortcut Files Security Feature Bypass with a CVSS of 8.1. The first one, the Windows SmartScreen Security Feature Bypass allowed malicious actors to inject their own code into SmartScreen to gain code execution, which could then lead to data exposure, lack of system availability, or both. The reason it was rated at only 7.6 is that for the attack to work the bad guys needed to send that targeted user a malicious file and convince them to open it.

The other 0-day permitted unauthenticated attackers to bypass displayed security checks by sending a specially crafted file to a targeted user but here again, the user would somehow need to be induced to take action by clicking on the link.

In any event, those two flaws that were being actively exploited have been fixed.

The five flaws deemed CRITICAL, in order of increasing criticality, are a Windows Hyper-V Denial of Service Vulnerability with a score of **6.5**, a Windows Pragmatic General Multicast (PGM) Remote Code Execution Vulnerability scoring **7.5**, Microsoft Dynamics Business Central / NAV Information Disclosure Vulnerability with a score of **8.0**, and finally the two biggies coming in at **9.8** each: To few people's surprise, we have another Microsoft Exchange Server Elevation of Privilege Vulnerability and an Microsoft Outlook Remote Code Execution Vulnerability.

A senior staff research engineer at Tenable said in a statement that this was very likely to be exploited and that Exploiting this vulnerability could result in the disclosure of a targeted user's NTLM version 2 hash, which could be relayed back to a vulnerable Exchange Server in an NTLM relay or pass-the-hash attack to allow the attacker to authenticate as the targeted user.

Believe it or not, last Tuesday's updates also fixed 15 – yes, I said 15 – remote code execution flaws in Microsoft WDAC OLE DB provider for SQL Server that an attacker could exploit by tricking an authenticated user into attempting to connect to a malicious SQL server via OLEDB. 15! Wow.

And rounding off the patch batch, as I mentioned, is a mitigation (not a total fix) for a 24-year-old fundamental design flaw – not an implementation flaw, a design flaw – in the DNSSEC spec which, had they known of it, bad guys could have used to exhaust the CPU resources of DNS servers to lock them up for up to 16 hours after receiving just a single DNS query packet. We'll be talking about this bullet that the Internet dodged before we wrap up today's podcast.

Closing the Loop

Ben / @ElasticSpoon

Hey Steve, Love the show. I hated the security courses I took in school but you make it way more interesting. I haven't missed a podcast from you in 3 years. My question was: I was recently going through my password vault and changing duplicate passwords. I encountered a lot of sites with length and symbol restrictions on passwords (for example, no passwords longer than 20 characters, disallowing certain symbols). My understanding of passwords is that they all get hashed to a standard length regardless of the input, so it can't be for storage space reasons. So why the restrictions? Even if I input an 8 character password the hash will end up being 128 bits, or whatever. I would love some insight because it makes no sense to me. Thanks, Yuri

So, Yuri, you're not alone. When you look at it closely, none of this really makes any sense to anyone. I think the biggest problem we have today is that there are no standards, no oversight and no regulation, and everything is opaque to the user. We hope that passwords stored on a site's server are hashed. But we don't know that. And assuming that they are hashed, we don't know whether they're being hashed on the client side in the user's browser or on the server side after they arrive in the clear over an encrypted connection.

A very very long time ago, back at the dawn of Internet identity authentication, someone might have been asked by a customer support person over the phone, to prove their identity by reading their password aloud. The person at the other end of the line would be looking at it on their screen to see whether it matched what they had on record. Of course, as we know, this could only be done if passwords were being stored in the server in the clear – as, indeed, they originally were. And in that instance, you can see why the use of special characters with names like "circumflex", "tilde", "pound", "ampersand", "back apostrophe" and "left curly brace" might be difficult for people at each end of the conversation to understand. So restricting the character set to a smaller common set of characters made sense. And we also know that in those very early days a web server and a web presence was often just a fancy graphical front end to an existing monstrous old school mainframe computer whose account password policies predated the Internet. So even after a web presence was placed in front of the old mainframe, its ancient password policies were pushed through to the user.

Today, we all hope that none of that remains. But if we've learned anything on this podcast it's to never discount the power and strength of inertia. Even if there is no longer any reason to impose password restrictions of any kind (other than minimum length), restrictions may still be in place today simply because they once were. And hopefully all consumers have learned the lesson to never disclose a password to anyone at any time for any reason. We see reminders from companies which are working to minimize fraud, explicitly stating that no employee will ever ask for a customer's password under any circumstances.

The gold standard for password processing is for JavaScript or WebAsm running on the client to perform the initial password qualification and hashing. Some minimum length should be enforced, all characters should be allowed, and requirements of needing different character families – upper and lower case, numbers and special symbols – also make sense. That will protect people from using "123456" or "password" as their password. And those minimal standards should be clearly posted whenever a new password is being provided.

Ideally, there should be a list of password requirements with a checkbox appearing in front of each requirement, dynamically checked as its stated requirement is met. And the "Submit" button should be grayed-out and disabled until all requirements have checkboxes. A password strength meter would be another nice touch.

Once something arrives at the server, high power systems on the server side can hash the living daylights out of whatever arrives before it is finally stored. But since we now live in an era where mass data storage has become incredibly inexpensive and where there's good reason to believe that major world powers are already recording Internet transactions in the hope of eventually being able to decrypt today's quantum-unsafe communications once the use of quantum computers becomes practical, there's a strong case to be made for having the user's client hash the qualifying password before it leaves their machine to traverse the Internet. So JavaScript or preferably WebAsm should be used to hash the user's qualifying input.

Much of that gold standard that I just described is user-facing and its presence or absence is obvious. And we rarely see it today. But it would be necessary to reverse engineer individual web applications if we wished to learn exactly how each one operates. Since avoiding the embarrassment of breaches and disclosures is in each company's best interest, and since none of the things I've described is at all difficult to deploy today, we can hope that the need to modernize the user's experience while improving their security will gradually overcome the inertia that will probably always be present.

Felipe Mafra / @fnmafra

Hello Steve! First of all I'd like to thank you and Leo for the great show! I'd like to bring you something very interesting that recently happened on this side of the border. The Canadian Industry Minister François-Philippe Champagne proudly tweeted on Feb. 8th that they are banning the importation, sale and use of hacking devices, such as Flipper Zero, that are being widely used for auto theft.

This is an attempt to respond to the huge increase of auto thefts here in Canada. Even if I believe it's good that the government is trying to address this issue, I found myself that rather than blocking the usage of such devices it would be better if the industry was required to make things right by design.

This pretty much aligns with last week's SN episode 960 regarding security on PLC's, as we see no commitment from those industries to make their products safe by design.

Anyways, I really wanted to share this with you and get your perspectives.
Thank you again for the show, looking forward to 999 and beyond.
Best regards, Felipe Mafra @fnmafra

In past years we've spent some time looking closely at the automotive remote key unlock problem. What we learned is that it is actually a difficult problem to solve fully and that the degree of success that has been obtained by automakers varies widely – some systems are lame and others are just about as good as can be.

And we've seen even very cleverly designed systems fall to ingenious attacks. Remember the one where a system was based on a forward rolling code that was created by a counter in the key fob being encrypted by a secret key and the result of that encryption was transmitted.

This would create a completely unpredictable sequence of codes. Every time the unlock button was pressed the counter would advance and the next code would be generated. And no code ever received the auto would be honored a second time. So anyone snooping and sniffing could only obtain code that had just been used and would no longer be useful.

So what did the super-clever hackers do? They created an active attack: When the user pressed the unlock button, the active attack device would itself receive the code while simultaneously emitting a jamming burst to prevent the automobile from receiving it. So the car would not unlock. Since that happens when we're too far away and it's not uncommon, the user would shrug and press the unlock button again. This time the active attacking device would receive the second code, emit another jamming burst to prevent the car from receiving the second code, then itself send the first code it had received to unlock the car.

But implementing this bit of subterfuge, the attacker is now in possession of a code that the key fob has issued but which the car has never seen – and it's the next key in the sequence from the last code that the car did receive. It's diabolically brilliant and I think it provides some sense for what automakers are up against.

From a theoretical security standpoint, the problem is that all of the communication is one-way – key fob to auto. The key fob is issuing a one-way assertion instead of a response to a challenge. What's needed to create a fully secure system would be for the key fob's unlock button to send a challenge request to the car. Upon receiving that request the car transmits a challenge in the form of an unpredictable value resulting from encrypting a counter. Again, the counter is monotonic, upward counting 128 bits and it will never repeat during the lifetime of the universe – let alone the lifetime of the car or its owner. So, upon receiving that unique challenge sent by the key fob, it encrypts that 128 bit challenge with its **own** secret key and sends the result back to the car. The car, knowing the secret kept by its key fobs, performs the same encryption and verifies that what was sent was correct. I cannot see any way for that system to be defeated. The car will never send the same challenge and the key will never return the same response. And no amount of recording that challenge and response dialog will inform an attacker of the proper responses to future challenges.

The only problem with this system is that now both endpoints need to contain transceivers capable of receiving **and** transmitting. So that adds some additional cost, though not much in production since both already contained radios anyway. But what this does mean is that a simple software upgrade will not, and cannot, solve the problem. I doubt it's possible to create a secure one-way system that's safe against an active attacker while still reliably unlocking the vehicle without unduly burdening its user. The system I've just described is not rocket science; it's what any crypto-savvy engineer would design, and since this problem is now well understood, I'd be surprised if next-generation systems which fix this once and for all were not already on and off the drawing board and headed into production. But that doesn't solve the problem which exists, and will continue to exist, for today's automobiles.

So now let's come back to Felipe's point about Canada's decision to ban the importation of devices such as the Flipper Zero. We know it doesn't solve the problem. But will it reduce the severity of the problem? Yeah, probably somewhat. Kits will spring up to allow people to build their own. Canada is a big place. There's nothing to prevent someone from firing up manufacturing and creating home-grown Flipper Zero's or their like. What we keep seeing is that low hanging fruit is the fruit that gets picked and eaten. And many people won't take the time or trouble to exert themselves to climb a tree to obtain the higher-hanging fruit. Hand'em a piece? Sure. Work for it? Perhaps later. So I would argue that making car theft even somewhat more difficult will likely be a good thing. And the Flipper Zero is, at best, a hacker's gadget. It's not as if it has huge non-hacker applications.

M-Scott / @mscott_xxxiii

Steve, I'm wondering about your thoughts. The cyber security community seems to bemoan the lack of security baked into the original internet design and ceaselessly encourages designers of new technology to bake in security from the get go. Several books I'm reading for a cyber and information warfare class suggest that government regulation to require security is the answer and should have been placed on the internet in the initial design.

However, I suspect, if security had been a mandate on day 1, the robust cyber community we have today would not exist. I see the internet as more of a wicked problem where solutions and problems emerge together, but can't be solved up front.

Your thoughts? Thanks for your continued service to the community!

I suppose that my first thought is that those writing such things may be too young to recall the way the world was when the Internet was being created. I'm not too young. I was there looking up at the IMP – the Interface Message Processor – a big imposing white cabinet sitting at Stanford's AI lab in 1972... as the thing known as Darpa Net was first coming to life. The problems these authors are lamenting not being designed-in from the start didn't exist before the Internet was created. It's the success of the Internet, and its applications, that created the problems, and thus the needs, we have today. Also, back then we didn't really even have crypto yet. It's nice to say "well, that should have been designed in from the start" – but it wasn't until four years later, in 1976, that Whit Diffie, Marty Hellman and Ralph Merkle invented public key crypto. And a huge gulf exists between writing an academic paper to outline a curious and possibly useful theoretical operation in cryptography, and designing any robust implementation into network protocols. No one knew how to do that back then, and we're still fumbling around finding mistakes in TLS decades later. And one other thing these authors may have missed in their wishful timeline is that the applications of cryptography were classified by the US Federal government as munitions. In 1991, nineteen years after the first IMPs were interconnected, Phil Zimmermann, PGP's author, had trouble with the legality of distributing PGP on the Internet.

Today, no one would argue that security should always be designed in from the start – and we're still not even doing that very well. We're still exposing unsecurable web interfaces to the public-facing Internet, and that's not the Internet's fault. So anyone who suggests that the Internet should have been designed with security built-in from the start would have to be unaware of the way the world was back when the Internet was actually first being built. Some of us were there.

Jonathan Haddock 🏴󠁧󠁢󠁥󠁮󠁧󠁿 / @joncojonathan

Hi Steve. Thanks for the podcast, having been listening from very early on. Was just listening to episode 961 and the section about Facebook telling users they entered an old password.

A downside of this practice is that an attacker knows the incorrect password was one that the person has used. Given the prevalence of password reuse, the attacker is then in possession of a confirmed previous password that could still work for that user elsewhere.

A middle ground would be to simply say when the password last changed. That way, the user experience is maintained but the previous validity of the password is not exposed.

Thanks again. Jonathan

I think that's a very good point. It's not **as** useful to tell a user that their password was recently changed, but that fact is why it could be useful to an attacker. If, instead, they are only told the date of their previous password change, then someone guessing their passwords would not receive confirmation when a previous password was correctly guessed. Something had been bothering me about that ever since I first encountered it... and Jonathan put his finger on it. There is a bit of information leakage from Facebook's practice.

Shawn Milochik / @ShawnMilo

And while we're sharing brilliant observations from our listeners, get a load of this one!

On the topic of the Passwordless/e-mail only login: I think the largest benefit to the companies is that this practically eliminates password sharing. It's one thing to give someone your password for a news or a streaming site. It's quite another to give them access to your e-mail -- and probably the ability to reset your bank password, among other things.

Indeed, and that's such a terrific point. Great observation, Shawn!

Lars Exeler / @LarsExeler

Hi Steve, love the show and just became a Club Twit member. I am wondering about TOPT as a second factor: So a password is the "something you know" factor. A TOTP is the "something you own" factor. My question: Isn't the TOTP just based on a lengthy secret (in the QR code)? So in my mind it's just a second password, with the convenience of presenting itself as 6 digits, because it is combined with the actual time. What I am getting wrong here?
Regards from Germany, Lars

A password is something you know. But the way to think of the TOTP, which is based upon a secret key, is that it's "transient and time-limited proof of something else that you own." And, as we know, a password can be used over and over. But in any properly designed TOTP-based authentication, the 6-digit token not only expires every 30 seconds, but each such token is invalidated after its first use. This means that even an immediate replay attack which still fits inside that 30-second expiration window will be denied. It's those distinctions that make the TOTP a very powerful and different second factor.

Gavin Lanoe / @GavinLanoe
(This is a long one, but it needs to be shared.)

Dear Steve,

I first came across GRC in the early 2000's for recovering harddisks and your Shields Up scanner just as I was at the start of a career in IT and IT Security. About a year ago I rediscovered you on the SecurityNow podcast and also discovered the wider TWiT network. The SecurityNow podcast over the last year is now the highlight of my week and even goes so far as to have a diary note in my calendar so I can try to listen to the creation live at what is late at night where I live. [Gavin lives in Germany.]

This week I discovered an issue with routers provided by one of our local ISP's and thought if there ever was a reason to drop you a message worthy of your time this was it. [And I think it's even worthy of our listener's time.]

Over my 30 or so years in IT I, (as I'm sure we all have), gathered a number of people who we look after for home user / family and friends IT support. I noticed over the last week or so some of my 400 or so 'private clients' were reporting strange issues. They reported to be getting warnings when browsing that sites were dangerous and were getting a lot of 404's, streaming services weren't working and connections to email services via IMAP (Secured with SSL) were not working, connection and certificate errors and such.

When I sat back and thought about it, all the users were on the same ISP and were all using the ISP provided router. (Side note: I usually recommend replacing the ISP router with something more substantial but when it's working fine or doesn't need anything better, like a single elderly home user with an iPad and nothing else, it's hard to justify the spend.)

I got access to one of the users' routers and after some investigation I found that the DNS servers listed just didn't look right. A traceroute showed the server was in the Netherlands and we are in Guernsey, Germany. The listed owner of the IP had nothing to do with the local ISP. Reverting the DNS Servers to the ISP's addresses resolved all issues. (Later I reset them to Quad9).

I also found that the ISP had left SNMP enabled on the WAN interface with no restrictions, and the username & password to gain access to the router were both "admin".

I have rotated through all my other customers that have the same router and am locking them down, checking the DNS servers, disabling SNMP and resetting the password. I may even go as far as replacing the routers in their entirety because they may have also been compromised in some other, yet undiscovered way.

I wrote to the ISP yesterday to explain the issue and received a call back today. It did take me some time to explain that any suitably skilled bad actor can connect to the routers using SNMP with the default credentials (or easily guessed community string) and reconfigure the router settings. The engineer I was speaking to had trouble comprehending how this was possible without the web management enabled on the WAN side, but he eventually understood. We contacted another customer while we were talking and he got to see the issue first hand and is taking this up with their security team internally.

Lord, I hope so!

My understanding of their reason for this configuration is: They want to be able to remotely connect to a customer's router to assist in fault finding [Ha! They won't have to look very far!]*, they have a script that connects to the routers via SNMP and enables web management on the WAN interface for a period of time and then it removes the web based access again.*

***They didn't consider** that the open SNMP management interface, with default credentials and an easy to guess community string could be exploited in this way.*

The engineer did seem quite disgruntled that I often replace the ISP provided routers but as I explained to him if it has a vulnerability on it I am going to replace without any consideration for their convenience of remote diagnosis.

Thanks Steve, I hope this may get a mention on the show, but regardless I really do look forward every week to your deep dives into the detail behind the week's security related news. Warm regards, Gavin from Guernsey.

Wow! Gavin, thank you for sharing that. Apparently this ISP believes that SNMP stands for "Simply Not My Problem" – but they would be wrong, and this deliberate unprotected SNMP exposure with default credentials is breathtaking. It is a horrific breach of security.

SNMP is a funky, non-user friendly, UDP-protocol based, very old original networking system which allows for both the remote over-the-wire monitoring and the control of networking gear. Unfortunately, it also hails from an era before security was being given the due that it deserves.

On the one hand, it's a good thing that this ISP has the wisdom to keep their customer's web management interfaces disabled by default. But that good intention is completely undone by their persistent exposure of the router's SNMP service. I'm still shaking my head over that one.

One remaining mystery is what was going on with the DNS rerouting. Given that most of today's DNS is still unencrypted in-the-clear UDP packets, this would be part of the requirement for site spoofing... but with browsers having become insistent upon the use of HTTPS, it's no longer possible to spoof a site over HTTP... so someone would need to be able to create web server certificates for the sites they wish to spoof. The fact that it's so widespread – across an ISP's many customers – tells us that it's not a targeted attack. That means that it's somehow about making money, and it's unclear how redirecting DNS makes anyone money.

In any event, thanks again, Gavin, for sharing this really interesting bit of horror!

Jeff Zellen / @JeffZellen

Steve: Listening to SN961 and thinking about the discussion on passwordless logins and using mail as a single factor. Is my recollection incorrect that email is generally transmitted in the clear? Can't some unscrupulous actor sniff that communication? I'm especially concerned if the email contains a link to the site. This provides all the necessary info to anyone who can view the email. A 6 digit code without any reference to the site where it should be used would be more secure as it is both out of band, but also obfuscated from the site. Of course this is all based on my possibly antiquated recollection of the lack of privacy in email.

The eMail transit encryption numbers vary. I saw one recent statistic that said that only about 50% of eMail is encrypted in transit using TLS. But Google's most recent stats for Gmail state that 98% to 99% of their eMail transit is encrypted. And it's a good feeling to have that. GRC's server fully supports all of the transit encryption standards so it will transact using TLS with the remote end when it's available, and it's a good feeling knowing that when Sue and Greg and I exchange anything, since our eMail clients are all connecting directly to GRC's eMail server, everything is always encrypted.

But Jeff's point about the possible lack of eMail encryption in transit is a good one, since eMail security is definitely lagging behind web security now that virtually everything has switched over to HTTPS. On the web, we would never even consider logging into a site that wasn't encrypted and authenticated using HTTPS. And even back when almost everything was still using HTTP, web sites would switch their connections over to HTTPS just to send the login page and receive the user's credentials.

By comparison to the standard set by today's web login with HTTPS, eMail security is sorely lacking. If we revisit the question of using eMail only to login to a site, where that eMail carries what is effectively a login link, or a 6 digit authentication token, what we've done is reduce the login security of the site to that of eMail – which today we would have to consider to be good but not great, and certainly not perfect.

Another difference worth highlighting is that a web browser's connection to the website it's logging on to is strictly and directly end-to-end. The browser has a domain name and it must receive an identity asserting certificate that is valid for that domain. But this has never been true for eMail. Although modern eMail does tend to be point-to-point, with a server at the sender's domain directly connecting to the server at the recipient's domain, that is not always nor necessarily true. eMail has always been a store and forward transport. Depending upon the configuration of the sender and receiver, eMail might make several hops from start to finish. And since the body of the eMail doesn't contain any of its own encryption, that eMail at rest is subject to eavesdropping.

Last week I coined the term "login accelerator" to more clearly delimit that value added by a password. And the whole point of the eMail loop authentication model is the elimination of the "something you know" factor. Yet without the "something you know" factor, the security of eMail loop authentication can be no better than the security of eMail... which falls short of the standard set by the web's direct end-to-end certificate identity verification with encryption.

As a result of this analysis, we might be inclined to discount the notion of eMail loop authentication as being inferior to web-based authentication with all of its fancy certificates and encryption ... except for one thing ... the ubiquitous "I forgot my password" get out of jail free link is still the party soiler. It is everywhere and its presence immediately reduces all of our much ballyhooed web security to the security of eMail – imperfect as it may be. Even without eMail loop authentication, if a bad guy can arrange to gain access to a user's eMail, they can go to any site where the user maintains an account, click the "I forgot my password" link, receive the password recovery link by intercepting their eMail, and do exactly the same thing as if that site was only using eMail loop authentication.

What this finally means is that all of our identity authentication login, either using all of the fancy web technology, or just a simple eMail loop, is not **actually** any more secure than eMail, and that simple eMail loop authentication without any password is just as secure as web authentication so long as web authentication includes an "I forgot my password" eMail loop bypass. So the notion of a password being nothing more than a login accelerator holds, and the world should be working to increase the security of eMail since it winds up being the linchpin for everything.

Alex Neihaus ☁️ and fediverse @alex@air11.social / @yobyot

> *Re: #961 and overlay networks. pfSense has a killer Tailscale integration (https://www.netgate.com/blog/tailscale-on-pfsense-software). With this running, you can set up pfSense to be an exit node, combining safe local access* ***=AND=*** *VPN-like-move-my-apparent-IP-address. That's useful for services that are tied to your location. Second, some cable providers' apps require UPnP, which you can (at your own risk) enable in pfSense-Tailscale (https://tailscale.com/kb/1181/firewalls).*

Given that I'm a pfSense shop (as Alex knows), and that TailScale integrates well with its FreeBSD UNIX, TailScale will certainly be a contender for a roaming overlay network. Thanks, Alex!

IcyvRan TocVuc / @IcyvRan
I believe that we've touched on this before but it bears repeating…

> *Hi Steve, Given your discussion of throwaway email addresses during the last Security Now podcast episode, I'd like to bring your attention to DuckDuckGo's Email Protection service. This free service is designed to safeguard users' email privacy by removing hidden trackers and enabling the creation of unlimited unique private email addresses on the fly. This feature allows users to maintain a distinct email address for each website they interact with.*
>
> *What makes this even more compelling is DuckDuckGo Email Protection's integration with Bitwarden. Bitwarden has included DuckDuckGo among its supported services, alongside other email forwarding services. This integration allows Bitwarden users to easily generate unique email addresses for each website they interact with. Take a look at: https://www.spreadprivacy.com/protect-your-inbox-with-duckduckgo-email-protection/ and https://bitwarden.com/blog/add-privacy-and-security-using-email-aliases-with-bitwarden/*

The Internet Dodged a Bullet

The vulnerability has been codenamed "KeyTrap", and if ever there was a need for responsible disclosure of a problem, this was that time. Fortunately, responsible disclosure is what it received.

What the German researchers who discovered this realized was that an aspect of the design of DNS, specifically the design of the secure capabilities of DNS, known as DNSSEC, could be used against any DNSSEC-capable DNS resolver to bring that resolver to its knees. The receipt of a single UDP DNS query packet could effectively take that DNS resolver offline for as many as 16 hours, pinning its processor at 100%, and effectively denying anyone else that server's DNS services. It would have therefore been possible to spray the Internet with these single packet DNS queries to effectively shutdown all DNS services across the entire globe. Servers could be rebooted once it was noticed that they had effectively hung, but if they again received another innocuous looking DNS query packet, which is their job after all, they would have gone down again. Eventually, of course, the world would have discovered what was bringing all of its DNS servers to its knees. But the outage could have been protracted and the damage to the world's economies could have been horrendous. So now you know why today's podcast is titled: *"The Internet Dodged a Bullet."* We should never underestimate how utterly dependent the world has become on the functioning of the Internet.

The detailed research paper describing this was just released publicly yesterday, though the problem has been known since late last year. Here's how the paper's Abstract describes what these German researchers discovered, to their horror:

Availability is a major concern in the design of DNSSEC. To ensure availability, DNSSEC follows Postel's Law [RFC1122], which reads: "Be liberal in what you accept, and conservative in what you send." Hence, nameservers should send not just one matching key for a record set, but all the relevant cryptographic material, in other words, all the keys for all the ciphers that they support and all the corresponding signatures. This ensures that validation succeeds, and hence availability, even if some of the DNSSEC keys are misconfigured, incorrect or correspond to unsupported ciphers.

We show that this design of DNSSEC is flawed. *By exploiting vulnerable recommendations in the DNSSEC standards, we develop a new class of DNSSEC-based algorithmic complexity attacks on DNS, which we dub KeyTrap attacks.* ***All popular DNS implementations and services are vulnerable.*** *With just a single DNS packet, the KeyTrap attacks lead to* ***a two million times*** *spike in CPU instruction count in vulnerable DNS resolvers, stalling some for as long as 16 hours. This devastating effect prompted major DNS vendors to refer to KeyTrap as* ***"the worst attack on DNS ever discovered".*** *Exploiting KeyTrap, an attacker could effectively disable Internet access in any system utilizing a DNSSEC-validating resolver.*

We disclosed KeyTrap to vendors and operators on November 2, 2023. We confidentially reported the vulnerabilities to a closed group of DNS experts, operators and developers from the industry. Since then, we have been working with all major vendors to mitigate KeyTrap, repeatedly discovering and assisting in closing weaknesses in proposed patches. Following our disclosure, an umbrella CVE has been assigned.

So, believe it or not, all that just actually happened, as they say, behind closed doors.

Google's Public DNS and Cloudflare were both vulnerable, as was the very popular and widely deployed BIND 9 DNS implementation, and it was the one that could be stalled for as long as 16 hours.

So… what happened?

As the researchers wrote, months before this all came to light publicly, all major implementations of DNS had already been quietly updated because had this gotten out it could have been used to bring the entire Internet down. With Microsoft's release of patches which included mitigations for this, and after waiting a week for them to be deployed, the problem is mostly resolved (of you'll pardon the pun).

I say "largely" because this is not a bug in an implementation and apparently even now, some DNS servers will still have their processors pinned, but they will still be able to answer other DNS queries function. It sounds as though thread or process priorities have been changed to prevent the starvation of competing queries. Characterizing this as a "big mess" would not be an exaggeration.

KeyTrap exploits a fundamental flaw in the design of DNSSEC which makes it possible to deliberately create a set of essentially legal but ultimately malicious DNSSEC response records which the receiving DNS server will be quite hard pressed to untangle. Once the researchers had realized that they were onto something big, they began exploring all of the many various ways DNS servers could be stressed. So they created a number of different attack scenarios.

I want to avoid getting too far into the weeds of the design and operation of DNSSEC, but at the same time I suspect that **this** podcast's audience will appreciate seeing a bit more of the detail so that the nature of the problem can be better appreciated. The problem is rooted in DNSSEC's provisions for the resolution of "key-tag" collisions, multiple keys and multiple signatures.

I'm going to quote three paragraphs from their research paper. But just sort of let it wash over you so that you'll get some sense for what's going on without worrying about understanding it in any detail. You will not be tested on your understanding of this.   Okay, so they explain:

We find that the flaws in the DNSSEC specification are rooted in the interaction of a number of recommendations that in combination can be exploited as a powerful attack vector:

Key-tag collisions: *First, DNSSEC allows for multiple cryptographic keys in a given DNS zone, for example during key-rollover or for multi-algorithm support. Consequently, when validating DNSSEC, DNS resolvers are required to identify a suitable cryptographic key to use for signature verification. DNSSEC uses key-tag values to differentiate between the keys, even if they are of the same zone and use the same cryptographic algorithm. The triple of (zone name, algorithm, key-tag) is added to each respective signature to ensure efficiency in key-signature matching. When validating a signature, resolvers check the signature header and select the key with the matching triple for validation. However, the triple is not necessarily unique: multiple different DNS keys can have an identical triple. This can be explained by the calculation of the values in the triple. The algorithm identifier results directly from the cipher* *used to create the signature and is identical for all keys generated with a given algorithm. DNSSEC mandates all keys used for validating signatures in a zone to be identified by the zone name. Consequently, all DNSSEC keys that may be considered for validation trivially share the same name. Since the collisions in algorithm-id and key name pairs are common, the key-tag is calculated with a pseudo-random arithmetic function over the key bits to provide a means to distinguish same-algorithm, same-name keys. Using an arithmetic function instead of a manually chosen identifier eases distributed key management for multiple parties in the same DNS zone; instead of coordinating key-tags to ensure uniqueness, the key-tag is automatically calculated. However, the space of potential tags is limited by the 16 bits in the keytag field. Key-tag collisions, while unlikely, can thus naturally occur in DNSSEC. This is explicitly stated in RFC4034, emphasizing that key-tags are not unique identifiers. As we show, colliding key-tags can be exploited to cause a resolver not to be able to identify a suitable key efficiently but to have to perform validations with all the available keys, inflicting computational effort during signature validation.*

To interrupt for a second, cryptographic keys are identified by tags, and those tags are automatically assigned to those keys. Work on DNSSEC began way back in the 1990's when the Internet's designers were still counting bits and were assigning only as many bits to any given field as would conceivably be necessary. Consequently, the tags being assigned to keys were, and are today, only 16 bits long. Since these very economical tags only have 16 bits, thus one of 64K possible values, inter-tag collisions, while unlikely, are possible. If DNSSEC were being designed today, tags would be the output of a collision-free cryptographic hash function and there would be no provision for resolving tag collisions because there would be none. The paragraph I just read said *"the key-tag is calculated with a pseudo-random arithmetic function"*, in other words, something simple from the '90's that scrambles and mixes the bits around but doesn't do much more. Consequently, servers need to consider that key-tags are not unique.

Okay, on to the next problem:

Multiple keys: *The DNSSEC specification mandates that a resolver must try all colliding keys until it finds a key that successfully validates the signature or all keys have been tried. This requirement is meant to ensure availability. Even if colliding keys occur, such that some keys may result in failed validation, the resolver must try validating with all the keys until a key is found that results in a successful validation. This ensures that the signed record remains valid and the corresponding resource therefore remains available. However, this "eager validation" can lead to heavy computational effort for the validating resolver, since the number of validations grows linearly with the number of colliding keys. For example, if a signature has ten colliding keys, all with identical algorithm identifiers, the resolver must conduct ten signature validations before concluding the signature is invalid. While colliding keys are rare in real-world operation, we show that records created to deliberately contain multiple colliding keys can be efficiently crafted by an adversary, imposing heavy computation upon a victim resolver.*

And, finally, the third and final problematic design feature:

Multiple signatures: *The philosophy of trying all of the cryptographic material available to ensure that the validation succeeds also applies to the validation of signatures.*

Creating multiple signatures for a given DNS record can happen, for example, during a key-rollover. The DNS server adds a signature with the new key, while retaining the old signature to ensure that some signature remains valid for all resolvers until the new key has propagated. Thus, parallel to the case of colliding keys, the RFCs specify that in the case of multiple signatures on the same record, a resolver should try all the signatures it received until it finds a valid signature or until all signatures have been tried.

Okay. So we have the essential design features which were put into the DNSSEC specification in a sane way with the purpose of never failing to find a valid key and signature for a zone record. Their term for this is "eager validation", and they write:

We combine these requirements for the eager validation of signatures and of keys, along with the colliding key-tags to develop powerful DNSSEC-based algorithmic complexity attacks on validating DNS resolvers. Our attacks allow ***a low-resource adversary*** *to fully DoS a DNS resolver for up to 16 hours* ***with a single DNS request.***

Members from the 31-participant task force of major operators, vendors and developers of DNS & DNSSEC, to which we disclosed our research, dubbed our attack: "the most devastating vulnerability ever found in DNSSEC."

The researchers devised a total of four different server-side resource exhaustion attacks and I was a little tempted to title today's podcast after three of them. Had I done so today's podcast would have been titled: "SigJam, LockCram & HashTrap." And while I certainly acknowledge that have been fun, I really didn't want to lose sight of the fact that the entire global Internet really did just dodge a bullet. And we don't know which foreign or domestic cyber intelligence services may today be silently saying *"Darnit!! They found it... that was one we were keeping in our back pocket for a rainy day, while keeping all of our foreign DNS server targeting packages updated."*

So what are these four different attacks?

SigJam utilizes an attack with one key and many signatures. They write:

The RFC advises that a resolver should try all signatures until a signature is found that can be validated with the DNSKEY(s). This can be exploited to construct an attack using many signatures that all point to the same DNSSEC key. Using the most impactful algorithm, an attacker can fit 340 signatures into a single DNS response, thereby causing 340 expensive cryptographic signature validation operations in the resolver until the resolution finally terminates by returning a SERVFAIL response to the client. The SigJam attack is thus constructed by leading the resolver to validate many invalid signatures on a DNS record using one DNS key.

The LockCram attack does the reverse, using many keys and a single signature. They write:

Following the design of SigJam we develop an attack vector we dub LockCram. It exploits the fact that resolvers are mandated to try all keys available for a signature until one validates or all have been tried. The LockCram attack is thus constructed by leading a resolver to validate one signature over a DNS record using many keys. For this, the attacker places multiple DNS keys in the zone which are all referenced by signature records using the same triple of (name, algorithm, key tag). This is not trivial, as resolvers can de-duplicate identical DNSKEY records *and their key tags need to be equal. A resolver that tries to authenticate a DNS record from the zone attempts to validate its signature. To achieve that, the resolver identifies all the DNSSEC keys for validating the signature, which, if correctly constructed, conform to the same key tag. An RFC-compliant resolver must try* ***all*** *the keys referred to by the invalid signature before concluding the signature is invalid, leading to numerous expensive public key cryptography operations in the resolver.*

The next attack "KeySigTrap" combines the two previous attacks by using multiple key and multiple signatures. They explain:

The KeySigTrap attack combines the many signatures of SigJam with the many colliding DNSKEYs of LockCram, creating an attack that leads to ***a quadratic increase*** *in the number of validations compared to the previous two attacks. The attacker creates a zone with many colliding keys and many signatures matching those keys. When the attacker now triggers resolution of the DNS record with the many signatures, the resolver will try the first key to validate all signatures. After all signatures have been tried, the resolver will move to the next key and again attempt validation with all signatures. This continues until all pairs of keys and signatures have been tried. Only after attempting validation on all possible combinations does the resolver conclude that the record cannot be validated and returns a SERVFAIL to the client.*

Elsewhere in their report when going into more detail about this KeySigTrap, they explain that they were able to set up a zonefile containing 582 colliding DNSSEC key and the same 340 signatures that we saw in SigJam. Since the poor DNS resolver that receives this response will be forced to test every key against every signature, that 582 times 340 or 197,880 expensive and slow public key cryptographic signature tests. And that was caused by sending that DNS server a single DNS query packet.

Interestingly, the researchers discovered that not all DNS servers were similarly affected. For some reason, several took the request much harder. Upon investigating they discovered why.

The Unbound DNS server is DoSed approximately 6 times longer than the other resolvers. The reason is the default re-query behavior of Unbound. In a default configuration, Unbound attempts to re-query the nameserver five times after failed validation of all signatures. Therefore, Unbound validates all attacker signatures 6 times before returning a SERVFAIL to the client. Essentially, Unbound is being penalized for being a good citizen. Disabling default re-queries brings Unbound back to parity with the other.

But BIND was the worst with the 16-hour DoS from a single packet. They explained:

Investigating the cause for this observation, we identified an inefficiency in the code, triggered by a large number of colliding DNSSEC keys. The routine responsible for identifying the next DNSSEC key to try against a signature does ***not*** *implement an efficient algorithm to select the next key from the remaining keys. Instead, it re-parses all keys again until it finds a key that has not been tried yet. This algorithm does not lead to inefficiencies in normal operation where there might be a small amount of colliding keys. But when many keys collide, the resolver spends a large amount of time parsing the keys to select the next key, which extends the total response duration of the DoS to 16h.*

What all of this should make clear is that these potential problems arise due to DNSSEC's deliberate "eager to validate" design. DNS servers really want to attempt all variations, which is exactly what gets them into trouble. The only solution will be something heuristic. We've talked about heuristics in the past. They can be thought of as a rule of thumb, and they usually appear when exact solutions are not available – which is exactly the case here.

As we might expect, the researchers have an extensive section of their paper devoted to "what to do about this mess," and they worked very closely for months with all of the major DNS system maintainers to best answer that question. I'll skip most of the blow-by-blow, this but give a sample solution. Under the subhead "Limiting all validations" they wrote:

The first working patch capable of protecting against all variants of our attack was implemented by Akamai. In addition to limiting key collisions to 4, and limiting cryptographic failures to 16, the patch also limits total validations in ANY requests to 8.

Evaluating the efficacy of the patch, we find the patched resolver does not lose any benign request even under attack with > 10 attacking requests per second. The load on the resolver does not increase to problematic levels under the ANY type attack with 10 req/s, and the resolver does not lose any benign traffic. It thus appears that the patch successfully protects against all variations of KeyTrap attacks.

Nevertheless, although these patches prevent packet loss, they still do not fully mitigate the increase in CPU instruction load during the attack. The reason that the mitigations do not fully prevent the effects of the KeyTrap attacks is rooted in the design philosophy of DNSSEC. Notice however that we are still closely working with the developers on testing the patches and their performance during attack and during normal operation.

The disclosure timeline for this is extra-interesting since it provides a good sense for the participants and the nature of their efforts and interactions:

In the following, we describe the timeline of disclosure to indicate how the vulnerability was reported and how we worked with the experts from industries to find solutions for the problems that we discovered.

- 02.11.2023: Initial disclosure to key figures in the DNS community. Both confirm that KeyTrap is a severe vulnerability, requiring a group of experts from industry to handle.

- 07.11.2023: Confidential disclosure to representatives of the largest DNS deployments and resolver implementations, including Quad9, Google Public DNS, BIND9, Unbound, PowerDNS, Knot, and Akamai. The group of experts agrees that this is a severe vulnerability that has the potential to cut off internet access to large parts of the Internet in case of malicious exploitation.

  A confidential chat-group is established with stakeholders from the DNS community, including developers, deployment specialists and the authors. The group is continuously expanded with additional experts from the industry to ensure every relevant party is included in the disclosure. Potential mitigations are discussed in the group.

- 09.11.2023: We share KeyTrap zonefiles to enable developers to reproduce the attack locally, facilitating the development of mitigations
- 13.11.2023: Akamai presents the first potential mitigation of KeyTrap by limiting the total number of validation failures to 32. (That doesn't work)
- 23.11.2023: Unbound presents its first patch, limiting cryptographic failures to a maximum of 16, without limiting collisions.
- 24.11.2023: BIND9 presents the first iteration of a patch that forbids any validation failures.
- 08.12.2023: An umbrella CVE-2023-50387, is assigned to the KeyTrap attacks.
- 02.01.2024: After discussions with developers, we find some have problems recreating the attack in a local setup. We thus provide them an updated environment with a DNS server to ease local setup and further facilitate testing of patches. (In other words, the researchers actually put this live on the Internet so that the DNS developers would DoS their own DNS servers.
- 03.01.2024: BIND9 presents the second iteration of a patch, limiting validation failures.
- 03.01.2024: Our newly implemented DS-hashing attack proves successful against all mitigations not limiting key collisions, including BIND9 and Unbound, and is disclosed to the group.
- 16.01.2024: Our ANYTYPE attack circumvents the protection from limiting colliding keys and limiting cryptographic failures.
- 24.01.2024: The first working patch is presented by Akamai. Other resolvers are implementing derivations of the countermeasures to protect against the attacks.

And so now we get to their final conclusions. They write:

*Our work revealed a fundamental design problem with DNS and DNSSEC: Strictly applying Postel's Law to the design of DNSSEC introduced a major and devastating vulnerability in virtually **all** DNS implementations. With just **one** maliciously crafted DNS packet an attacker could stall almost any resolver, for example the most popular one, BIND9, for as long as 16 hours.*

The impact of KeyTrap is far reaching. DNS evolved into a fundamental system in the Internet that underlies a wide range of applications and facilitates new and emerging technologies. Measurements by APNIC show that in December 2023, 31.47% of web clients worldwide were using DNSSEC-validating resolvers. Therefore, our KeyTrap attacks have effects not only on DNS but also on any application using it. An unavailability of DNS may not only prevent access to content, but risks also disabling security mechanisms, like anti-spam defenses, Public Key Infrastructure (PKI), or even inter-domain routing security like RPKI or rover.

Since the initial disclosure of the vulnerabilities, we've been working with all major vendors on mitigating the problems in their implementations, but it seems that completely preventing the attacks will require fundamentally reconsidering the underlying design philosophy of DNSSEC, in other words, to revise the DNSSEC standards.

https://www.athene-center.de/fileadmin/content/PDF/Technical_Report_KeyTrap.pdf

So, as we can see, as I titled this podcast, The Internet really did dodge a bullet.

It is also terrific to see that at the technical and operational level we have the ability to quickly gather and work together to resolve any serious trouble that may arise.

Fortunately, that doesn't happen often.